SANDIA REPORT

SAND2014-20421
Unlimited Release
Printed December 2014

A Workflow for Parameter Calibration and and Model Validationin SST/Macro: Interim Report

Philippe Pébay, Jeremy Wilke, and Khachik Sargsyan

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2014-20421
Unlimited Release
Printed December 2014

A Workflow for Parameter Calibration and and Model Validationin SST/Macro: Interim Report

Philippe Pébay
Sandia National Laboratories
M.S. 9159, P.O. Box 969
Livermore, CA 94551, U.S.A.
pppebay@sandia.gov

Jeremy Wilke
Sandia National Laboratories
M.S. 9159, P.O. Box 969
Livermore, CA 94551, U.S.A.
jjwilke@sandia.gov

Khachik Sargsyan
Sandia National Laboratories
M.S. 9051, P.O. Box 969
Livermore, CA 94551, U.S.A.
ksargsy@sandia.gov

Abstract

This brief report explains the method used for parameter calibration and model validation in SST/Macro and the set of tools and workflow developed for this purpose.

Acknowledgments

The authors would like to thank Cosmin Safta (Sandia National Laboratories) for his help with cross-platform building of UQTk.

Contents

This page intentionally left blank

1 Introduction

UQTk relies on Bayesian inference methods for model parameter calibration. Model validation is a direct result of calibration post-processing. Specifically, a model is considered validated if the calibrated model parameters and the associated uncertainties can well explain or predict the available data. The library allows for implementation of Bayesian calibration using Markov chain Monte Carlo (MCMC) methods.

The MCMC technique essentially searches the parameter space and compares model results with available data. However, each parameter sample invokes a model evaluation, and as a result the MCMC approach requires that many samples be processed for properly estimating the uncertainties, which can quickly become computationnally too costly. In such cases, our modeling approach is to replace the full model, viewed as a black-box, by a surrogate, again as a black-box itself, that is constructed by the forward UQ techniques also available in UQTk.

In the following sections we describe the set of tools which we have developed in order to implement this approach and we briefly illustrate their use for an examplar case.

2 Pre-requisites

In order to run the UQ workflow which is described hereafter, you need to have SST/Macro and UQTk installed on your system. You will also need Python.

2.1 Python

The scripts provided with the UQ workflow are known to work with Python 2.7.3.

The following Python modules are needed: `os`, `sys`, `getopt`, `subprocess`, `shutil`, `math`, `random`, `array`, `numpy`, `scipy`, and `cPickle`.

2.2 SST/Macro

The UQ worfklow is known to work with a build of change set `2925:75b84bad7398` of SST/Macro. No particular configuration options are required.

2.3 UQTk

The UQ workflow is known to work with dynamically linked libraries built and installed from commit `17eb4b9b177bbb29f182e7697331feb1e6a6bd28` of UQTk.

You will need to build and UQTk with it Python bindings (PyUQTk turned ON).

3 Method and Tools

3.1 Overview

The parameter calibration worflow is decomposed in the following steps, which can be regrouped in two main phases:

1. The first phase is the one in which the surrogate models are generated, using the following scheme:
   (a) A pre-processing step whose aim is to generate a set of quadrature points across the multi-dimensional parameter space. Specifically, a set of quadrature points are generated using a projection-based methodology available in UQTk which samples the Cartesian product of the support of the initial uniform prior distributions for the parameters to be inferred.
   (b) A main processing step, in which a SST/Macro simulation per each of the aforementioned quadrature points is executed. This is the most computationnaly costly part of the workflow, but it lends itself to an embarrasingly parallel implementation.
   (c) A post-processing step of the SST/Macro simulations, whereby as many surrogate models as there are experimental observations are generated, in the form of polynomial chaos expansions.
2. In the second stage, the parameter calibration *per se* is performed with MCMC scheme as follows:
   (a) A pre-processing step where the surrogate models output are retrieved for the polynomial chaos representations in a form that is suitable to the MCMC implementation.
   (b) A main processing step, which performs MCMC inference of the distribution of the parameters to be calibrated, using the experimental data generated by runs of the test application that corresponds to the simulated application used in the main step of the first phase.
   (c) A post-processing step in which, in order to obtain a reliable Markov chains, the originally computed ones are "thinned" by the means of user-defined origin and stride control parameters.
3. Last, the worflow also provides subsequent analysis steps which extracts various quantities of interest, such as acceptance rate and maximum *a posterior* estimates of the inference parameters. It also re-scales to their original physical scale all obtained quantities which are normalized in the computational process.

3.2 Tools

The tarball provides the following extensions to UQTk:

- `uq_pc.py`: A Python wrapper to call the relevant pre and post-processing steps of the first phase, whose corresponding methods are provided in UQTk.
- `model.py`: A Python file providing a helper class and methods for the above wrapper.
- `rescale.x`: A Bash script to un-scale quantities contained in an input stream from a physical scale to normalized $[-1, 1]$ intervals, as well as to provide the converse operation given an input physical state.

In addition, the tarball provides the following Python scripts in order to implement the scheme explained in §3.1:

- `extract_matrix.py`: A Python script to extract the estimated times, as simulated by the main step of the first phase of the work flow, and transform them into a matrix form suitable for the post-processing step using `uq_pc.py`.
- `generate_hypercube.py`: A small convenience Python script to generate the normalized hypercube (with dimension equal to the number of calibration parameters) used as input by the MCMC engine, and in the form expected by it.
- `extract_PC.py`: A Python script providing the glue between the first and second stages of the workflow. Specifically, it extracts those data in the surrogate model output by the first phase that are needed by the parameter inference scheme and prepares them in the form that is expected by the MCMC code.
- `thin_chain.py`: A Python script to "thin" a computed Markov chain, given a starting point and a stride. For instance, one can decide to retain only one in 10 computed points within the last half of the chain.
- `workflow.sh`: A shell script that assembles all the above, along with a number of shell commands, to implement the workflow described in §3.1. This script has a number of control parameters which are described below.

Note that at this point, all steps of the scheme described in §3.1 are implemented, except for the parameter sweep of calling SST/Macro simulations over the set of quadrature points, because this step is entirely defined by the nature of experiment of interest. As a result and for the sake of generality, we decided to not provide a default parameter sweep implementation as the commonalities between conceivable experiments are essentially reduced to nil. Instead, we provide an exemplar Python script in §4, which can be either customized for experiments close to that explained here, or used as a template to devise entirely different experiments.

3.3 Control Parameters

The workflow script has the following input parameters:

P_DOMAIN : The name of a file providing the support of the uniform prior for each calibration parameter. This file must have 2 columns (lower and upper bounds) and as many rows as there are calibration parameters.

X_DATA : The name of a file containing the design parameters over which the experimental computations where executed. The number of columns is equal to the number of design parameters and the number of rows is equal to the number of individual experiments.

Y_DATA : The name of a file containing the results of the experimental computations where executed. The number of columns is equal to the 1 as only one result per experiment is considered (execution time). and the number of rows is equal to the number of individual experiments.

N_QUAD_PER_DIM : Number of quadrature points per dimension.

N_VALIDATION : Number of additional validation points, to be added to the set of quadrature points.

DIM : Dimensionality of the problem, *i.e.*, the number of calibration (inference) parameters.

ORDER : Order of the polynomial chaos surrogates.

N_OBS : Number of individual experiments (observations).

N_THIN : Starting point, expresssed as a fraction of the entire chain length, for the thinning process of the computed Markov chain.

R_THIN : Fraction of the computed Markov chain to be retained in the thinning process, beginning after the starting point.

N_STEPS : Number of steps in the MCMC inference.

GAMMA : Acceptance rat for the MCMC scheme.

Note that the union of all control parameters of each of the individual components contained in the workflow is substantially richer than this list. However, only those control parameters which we found useful during our first phase of experiments are exposed here. It is possible, and even likely, that subsequent work will result in more control parameters be available.

3.4 Output

The workflow is executed as follows:

```
./workflow.sh
```

and it prints information to standard output about its current stage until it completes. The last printed values are those of the the MAP estimates of the inference parameters.

In addition, the workflow creates the following output data files:

`accept_rate.dat` : Containing the MCMC acceptance rates computed every 5%The full computed Markov chain, with normalized values for the inference parameters.

`chain.dat` : The full computed Markov chain, with normalized values for the inference parameters.

`thin_chain.dat` : The "thinned" subchain extracted from the above.

`MAPparams.dat` : Containing the MAP estimates of the normalized inference parameters, as well as that of the variance.

`MAPparams_sc.dat` : Containing the MAP estimates of the inference parameters, rescaled to their corresponding original physical scales.

`allsens.dat` : Containing the sensitivy contributions to each of the design parameters in normalized scale, for each individual experiment.

`allsens_sc.dat` : Containing the sensitivy contributions to each of the design parameters in physical scale, for each individual experiment.

4 Example

We provide here a detailed example to illustrate how the abovementioned tool set can be used for model calibration and validation of SST/Macro simulations.

4.1 Skeleton Application

The tarball provides the skeleton application in the form of C++ programm making MPI calls which will be intercepted by SST/Macro. This skeleton application is contained in a file called `main.cc` and is provided along with a `Makefile` so that it suffices to execute the following command:

```
make
```

in order to build the skeleton application and link it against the SST/Macro libraries. As a result, an executable called `runsst` will be created.

In the considered examplar application, 2 design parameters are used, so that the Cartesian product of their sets of possible values provide the set of experiments whose results can be either experimental (calibration data) when obtained by running the test application on a real platform while executing all paralllel communications, or simulated (inference data) when obtained with SST/Macro simulations which capture MPI calls and estimate their corresponding execution time given a hardware architecture otherwise fixed (*i.e.*, the sweep is performed while only the calibration parameters vary in the SST/Macro input decks).

4.2 Calibration Parameters

The exemplar case provided in the tarball is 3-dimensional, as the calibration (inference) parameters represent, respectively: network bandwith (expressed in Bytes per second), network hop latency (expressed in seconds), and injection latency (expressed in seconds too).

Therefore, in order to complete the implementation of the workflow of §3.1 for this examplar case, the tarball also contains a Python script, called `parameter_sweep.py`, which provides the SST/Macro parameter sweep driver of the design parameters, across the pre-processed quadrature points, for a handful of possible otherwise constant hardware configurations (in SST/Macro input deck format).

4.3 Experimental Data

The tarball also contains a subdirectory `data_example` in which the files `rank_size_8x127.dat` and `time_8x127.dat` can be found. These files respectively provide the input parameters and

corresponding execution times of the non-skeletonized application (thus with actual MPI calls) measured on a real parallel system.

Specifically, the input parameters are, first, the computational node index (each node corresponding to a unique set of possible paths through the network from the initiating node, considered as node 0) and, second, the message size sent my MPI by the application. There are 127 computational nodes and 8 different sizes, resulting in 1016 lines in each of these files, viewed as 1016 independent experiments.

In addition, the experimental data directory contains a file call `pdomain_3d.dat`, which provides example bounds for the support of the uniform priors used for each of 3 inference parameters used in this exemplar case. These values are provided only for the sake of illustration but can me modified as desired; however, should a different input file be created, its number of columns must obviously match the number of columns of inference parameters.

4.4 Results

With the following workflow input parameters (which are those provided in the tarball):

```
P_DOMAIN="./data_example/pdomain_3d.dat"
X_DATA="./data_example/rank_size_8x127.dat"
Y_DATA="./data_example/time_8x127.dat"
N_QUAD_PER_DIM=4
N_VALIDATION=0
DIM=3
ORDER=3
N_OBS=1016
N_THIN=0.5
R_THIN=0.01
N_STEPS=100000
GAMMA=0.1
```

the workflow outputs the following maximum *a posteriori* (MAP) estimates for the 3 inference parameters, respectively:

```
4.023965718024637e+08 7.195485857933070e-08 9.546343308896442e-07
```

i.e., approximately 0.41GB/s for the network bandwith, 72ns for the hop latency, and 0.95μs for the injection latency. The corresponding values expected from expert knowledge are 0.38GB/s for the network bandwith, 100ns for the network latency, and 1–2μs.

5 Future Work

This brief report discussed the status of ongoing work at the time of printing. None of the material provided above is intended to remain in the fixed form, not even the workflow would can evolve substantially and be modified to allow for different experiments.

In particular, our findings so far indicate that interesting directions for future research include in particular the following:

- Perform inference with the plain model without invoking surrogates, *i.e.* eliminate the first stage of the workflow, and instead execute SST/Macro simulations within the computing step of the second stage.
- Remove non-important parameters from inference (which appears to be the case for example for the hop latency in our examplar case), assuming the input priors were chosen correctly.
- Do cross-validation would be useful, *i.e.*, hide some data, train with the rest and validate with the hidden data.

DISTRIBUTION:

2	MS 9018	Central Technical Files, 8944
1	MS 0899	Technical Library, 9536

Sandia National Laboratories